DOCUMENT RESUME

ED 281 498 IR 012 615

AUTHOR Collery, Michael T.
TITLE Color in Three-Dimensional Shaded Computer Graphics and Animation.
PUB DATE 85
NOTE 50p.; Master's Thesis, Ohio State University. Illustrations may not reproduce.
PUB TYPE Dissertations/Theses - Master Theses (042)

EDRS PRICE MF01/PC02 Plus Postage.
DESCRIPTORS Aesthetic Values; Algorithms; *Animation; Color; *Computer Graphics; *Light; Models; Programing; Three Dimensional Aids; Visual Arts
IDENTIFIERS *Color Theories; Gray Scales; Polygons

ABSTRACT

Directed primarily toward the person familiar with computer technology rather than the traditional art educator, this master's thesis addresses new approaches to color theory and its impact on electronic art media. Technical considerations related to color coming from electronic sources, i.e., radiant light, and differences between radiant and reflected light as explained by traditional color theories are discussed, as well as techniques for producing various effects with computer graphics. The thesis is divided into 10 chapters: (1) Introduction; (2) Color Theories; (3) Gray Scales; (4) The Limits of Color; (5) Polygonal Objects; (6) Display Algorithms; (7) Framebuffer Colormap Configurations; (8) Polygon Colors and Vertex Colors; (9) Creating Color Files; and (10) Conclusion. The text is illustrated with 25 figures, and 14 references are listed. (MES)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED281498

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

X This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality.

• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

COLOR IN THREE-DIMENSIONAL SHADED COMPUTER GRAPHICS AND ANIMATION

A Thesis

Presented in Partial Fulfillment of the Requirements
for the Degree Master of Arts

by

Michael T. Collery, B.F.A.

The Ohio State University
1985

Approved by

Thomas E. Linehan

Adviser
Department of Art Education

Copyright (c) 1985
by Michael T. Collery
All rights reserved.

IR012615

BEST COPY AVAILABLE

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY

Michael T. Collery

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

ACKNOWLEDGMENTS

I would like to thank Charles Csuri for giving me a chance to make good, and my advisor Dr. Thomas Linehan whom I have worked with for more years then both of us would like to admit. I would like to thank the "old gang" at the Computer Graphics Research Group who taught me so much and kept me company during those long days and nights. A special thanks to my wife, Linda Martin, for putting up with my many hours of absence and finally to those at Cranston/Csuri Productions, who have given me an opportunity to but my ideas to work.

Table of Contents

Acknowledgements ii
Table of Figures iv
Chapter 1. INTRODUCTION 1
Chapter 2. COLOR THEORIES 3
Chapter 3. GRAY SCALES 5
Chapter 4. THE LIMITS OF COLOR 9
Chapter 5. POLYGONAL OBJECTS 12
Chapter 6. DISPLAY ALGORITHMS 17
Chapter 7. FRAMEBUFFER COLORMAP CONFIGURATIONS 19
Chapter 8. POLYGON COLORS AND VERTEX COLORS 29
Chapter 9. CREATING COLOR FILES 32
Chapter 10. CONCLUSION 40
List of References 44

List of Figures

Figure 1. Additive Color 2
Figure 2. Maxwell Color Triangle 3
Figure 3. Typical Grayscale 5
Figure 4. Triangle of color-receiver primaries plotted in CIE color space 10
Figure 5. Flying Logos 12
Figure 6. Wineglasses with Fake Reflections 13
Figure 7. Don Stredney's and Chris Steele's Brain 14
Figure 8. Duplicating and Combining Polygonal Data 15
Figure 9. Faceted and Smooth Shaded Object with Highlight 17
Figure 10. Parallel Color Lookup Table Configuration 19
Figure 11. Full Color Framebuffer Configuration 20
Figure 12. Anti-aliased and Aliased Edges 22
Figure 13. Transparent Bubbles 24
Figure 14. Bowling Pins 25
Figure 15. Colormap Changes 27
Figure 16. Rebirth 29
Figure 17. Looped Vertex Colors on Solids of Revolution 30
Figure 18. Strips vs Checkerboard Effect 32
Figure 19. Polygonal Structure vs Polygon Color Pattern 33
Figure 20. Polygonal Structure vs Vertex Color Pattern 34
Figure 21. Smoothing Vertex Color Blending 36
Figure 22. Landscape with Vertex Coloring 37
Figure 23. Don Stredney's Eyeball 39
Figure 24. Texture Mapped Images 40
Figure 25. Metallic Effect 42

CHAPTER 1

INTRODUCTION

This thesis deals with new approaches to color theory and its impact on the electronic art media. Increasing numbers of artists are becoming involved with computer graphics and animation. They are discovering that certain aspects of color theories used in traditional art media are not applicable to computer graphics. This research is an attempt to identify important technical considerations relative to color coming from electronic sources, which is radiant light and not reflected light as explained by traditional color theories. The material presented here is directed toward the person who has a familiarity with computer technology rather than the traditional art educator. However, this information may have some important implications for those teaching color theory in art education.

Color in computer graphics is a wide and far reaching topic. Exactly where the domain of color begins and ends in computer graphics is hard to define. In a raster image, (like a television image) every "pixel" or picture element on the screen is little more than a speck of colored light. The field of computer graphics is becoming more visually oriented and less rooted in mathematics and physics. The overall affect of an image (or a series of images in the case of animation) is becoming more important than how the image was created. Past emphasis on scientific and mathematical modeling is giving way to methods of creating images based on the look of the final product. Researchers are still drawing on mathematics and science to achieve their results, but many are incorporating diverse sources of knowledge, including the arts and the social sciences, to research materials. "Fake" effects and "real" effects are becoming indistinguishable. A key issue in evaluating the image is its aesthetic quality. A refined use of color is therefore becoming increasingly important. The ability to control the colors on the screen largely determines the quality of an image.

The color of a pixel on the screen can be determined by a large number of processes. These include digital painting, image digitization, solid modeling, and screen space image manipulations. Often an image will be created by the use of numerous digital image synthesis and image

processing techniques. These approaches can be integrated with each other. One instance is the mapping of a scanned image onto a computer-generated, three-dimensional object. From a creative point of view, the more interaction between approaches the more flexible the image generation environment.

My personal experiences in computer graphics have primarily been with solid modeling systems based on polygonal shape representation. Therefore my discussion of color in computer graphics will be focused in this direction. Much of the information in this paper will be applicable to computer graphics in general. When dealing with system specific examples I will make an effort to explain these systems, so that a clearer understanding of computer graphics is gained by artists and art educators. I will review some of the literature available on color, and relate this literature to color as it exists in color spaces created by computer graphics hardware and software.

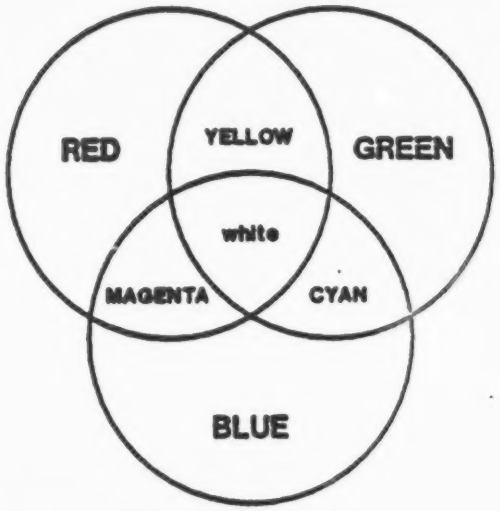


Figure 1.
Additive Color

CHAPTER 2

COLOR THEORIES

The major feature of color mixing on a computer is that it follows the laws of additive color theory, which defines the rules of color that light itself adheres to. In many ways this is the complete opposite of subtractive color. The color of an object that does not emit its own light is determined by those frequencies of the visual spectrum that are absorbed and reflected by the object. The rays that are reflected are perceived by the human eye as the color of the object. Therefore, under the substractive theory, if all pigments are mixed together the result will be black due to the fact that all light is absorbed. With additive mixing a white color results from combining every color of light.

In additive color there are only three primaries. These primaries are red, green, and blue. These primaries are often abbreviated in computer graphics and electronics literature and are referred to as RGB color. From these three colors all other colors of light can be obtained.

The number of primaries in subtractive color theory varies with each theoretician. Albers uses three primaries: red, yellow and blue, in his discussions of color theory. Other subtractive color theorists such as Ostwald and Birren, describe four primary colors. These are red, yellow, green, and blue. The Munsell color system includes purple as a fifth primary color.

Often subtractive color space is represented as a circle with the complement positioned directly across the centroid from the primary. A two dimensional figure used to represent additive color space is known as a Maxwell color triangle. The complement of blue is found directly across from it, on the midpoint of the line connecting red and green. This color is yellow.

Despite the differences in additive and subtractive color, there are many similarities. This is due to the fact that human perception is more responsible for the way we see color than the actual physical stimuli. Therefore most concepts that hold true for subtractive color will also be true for additive color. The major difference between the two systems is in the way colors are mixed and

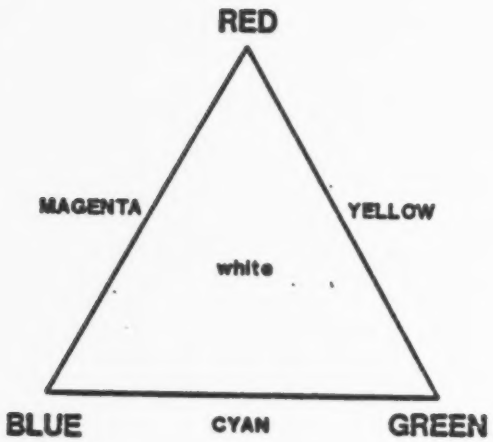


Figure 2.
Maxwell Color Triangle

not in the way they are perceived by the eye. For this reason, it is valuable for those involved in creating color images with a computer to acquaint themselves with texts on color theory, even though these books may be geared toward those who are working with pigments.

CHAPTER 3

GRAY SCALES

The gray scale is the foundation of color theory. This scale forms the basis for ideas about color scales and color harmony in general. The gray scale is the ground upon which color spaces are created. The central point in almost all color spaces is middle gray. The extremes are white and black.

Color theorists such as Birren [2] and Albers [1] are familiar with a phenomenon of human vision now known as the Weber-Fechner Law. This law states that the human eye's response to changes in light intensity is logarithmic rather than linear in progression. This means that if a light or color's intensity is increased by a linear value the eye/brain will not perceive the change as uniform. It will take an increase of double the original intensity to get a single step up the gray scale. A number series to represent this concept would look like 1,2,4,8,16,32,64,128,256 rather then 1,32,64,96,128,160,192,224,256. Understanding this law is very important, however it is generally overlooked by those involved in computer graphics.

Ed Catmull [7] exhibited knowledge of this law in his article on gamma compensation tables for correcting for the nonlinear response of color monitors. However he did not emphasize the importance of the Weber-Fechner Law in the light source calculation in three dimensional image synthesis, or electronic image generation in general.

In three dimensional image synthesis the shading of an object in the scene is calculated mathematically based on an equation known as the "lighting model". This model includes one or more mathematically defined light sources. The side of the object facing the light source is bright, the other side is dark. This type of shading is essential to the perception of depth in flat art work and is well known to the visual artist and color theorist. Newman and Sproull [11] explain several shading models in their book. These calculations are straight-forward and are well known throughout the graphics field. They will be described briefly in this thesis in a latter section.

Figure 3.
Typical Grayscale

The logical place to compensate for the Fechner-Webner Law would be during the shading model calculation. It is at this point that the color of a point on the screen is computed, based on the color assigned to the three dimensional data (object) being displayed and its relationship to the light source. Since this is where the shading from dark to light (the value scale) is computed, this is where the Weber-Fechner law should be considered.

However, the calculation of the lighting model is one of the most expensive steps in generating shaded graphics in that it is generally computationally intensive. These lighting models can become extremely complex, thus stimulating the rich visual reality in which we live. But the more complex the model is, the more machine time is required for its implementation. It often requires many hours to generate a single frame when using a complex lighting model, such as one with several light sources. It is for this reason that those involved with electronic image synthesis have to be concerned with reducing the

time required to calculate an image. Therefore the inclusion of a logarithmic function (which is computationlly intensive) as opposed to a simple linear function is a costly proposition and could drastically affect the time that it takes to compute an image. One way of getting around this is to calculate the logarithmic function only once for an entire image rather than on a pixel by pixel basis. This would result in a considerable saving of computer time.

This is made possible by using methods similar to those used by Catmull [7] and others known as gamma compensation. This calculation corrects for the fact that the output from color monitors is nonlinear. This equation straightens the response curve of linear changes in voltage within the monitor. This is done once to the "colormap" or "color lookup table" of the digital framebuffer, and affects the entire image instantaneously. A framebuffer is a special piece of computer hardware used to store digital images. It consists of a large mass of computer memory and digital to analog converters that translate the numbers stored in this memory to electronic signals. These signals are then used to control the red, green, and blue color guns within a television set or color monitor. Colormaps, lookup tables, and framebuffers will be discussed elsewhere in this thesis at some length. The important point here is that the ability to obtain a full range of values with uniform increments (gray scale) is a major consideration when working with shaded computer graphics, and that the values that make up an image can be altered instantly by a single write to the framebuffer lookup table.

I have used trigonometric curves to define color lookup tables that create reasonable looking images with a full range of value levels. These gray scales were developed to "look correct" on the monitor itself rather then on film, which is what standard gamma compensation is geared towards. This "eyeballing" approach is valid when the final product is to be recorded directly onto videotape without the use of a camera or film. In this case the image as it appears on a properly adjusted television monitor is the ultimate test of quality. If the artist is interested in recording their work onto videotape, it is important to develop a compensation table that creates a good looking image when the image is converted from a RGB to a NTSC signal (Nation Television Standard Code). A compensation table geared toward film output will not work when going directly to videotape. The result will be washed out colors.

If the artist is interested in having their work produced on film, gamma needs to be reconsidered in the more traditional approach. This is due to the nonlinear response of RGB monitors and the unique response curves of various film stocks. Film stocks vary in their sensitivity to light levels and color, therefore a unique gamma compensation curve needs to be computed for each film stock and each monitor in use.

CHAPTER 4

THE LIMITS OF COLOR

The range of color that any particular computer graphics system can produce is defined by several factors. System manufacturers often advertise that their computer display has a palette of 256 colors or 512 colors, or even larger numbers. These numbers are the result of the display hardware, and are directly related to the size of the bit field (amount of computer memory) used to store and transfer the digital color information to the analog output device. The relationship is straight-forward. The number of colors equals two raised to the power of the size of the bit field. A system with a one bit framebuffer will be able to create two colors or a black and white image without any shades of gray. A system with a two bit framebuffer will have a palette of four colors or a gray scale with four steps. The average computer video game has a palette of sixteen colors because it uses a four bit framebuffer. A standard size framebuffer is eight bits "wide" and has a palette of 256 colors.

It was stated earlier in this paper that there are three additive color primaries, red, green, and blue. From these colors all other colors can be produced. All colors on computer screens are created by mixing different intensity levels of red, green and blue. For example, a saturated bright yellow can be created by mixing full intensity red with full intensity green. A darker yellow can be created by mixing red and green of lower intensity or value. By mixing a high intensity red with a lower intensity green an orange color can be created. The size of the bit field defines the degree of similarity that can exist between two adjacent colors, or the subtlety of the palette. That is, a larger bit field allows more levels of color, and hence adjacent colors are more similar. This is analogous to adding the number 1 to the number 8, as compared to adding 1 to the number 64. The former results in a much larger percentage of change.

One technique that has been developed to broaden the range of color on the screen without lengthening the bit field is called "dithering" which is similar to silk screen printing, or Pointillist painting. One pixel on the screen will be colored red, while the one next to it will be colored green. If the viewer is far enough away

from the screen and unable to distinguish individual pixels, this pattern of red and green alternating pixels will be mixed in the mind of the viewer as yellow. This visual mixing is known as the Bezold Effect and is discussed by color theorists such as Albers [1]. A saturated red of full intensity can be worked into a pattern with black or white to produce various shades of red. Due to this visual perception phenomenon the range of color on the screen at any given time can be greatly extended past the physical limits of the display hardware.

With the drastic cuts in the price of computer memory due to advances in technology, the limitations caused by display hardware will play a lesser role in the image synthesis process. On framebuffers with larger bit fields (twenty four bits and over) the number of colors displayable on the screen goes into the millions. It is at this point that other types of limitations become apparent. One such limitation is the range of color possible on the television monitor itself. This is a subset [10] of the total visual spectrum, therefore certain colors can never be achieved with computer graphics despite advances in digital hardware. The color gamut of a digital image is restricted by its analogue display device. Some of the restrictions of the color monitor can be eliminated with film recorders that use filters in front of black and white monitors to produce color. However, film stocks are also subject to their own color limits.

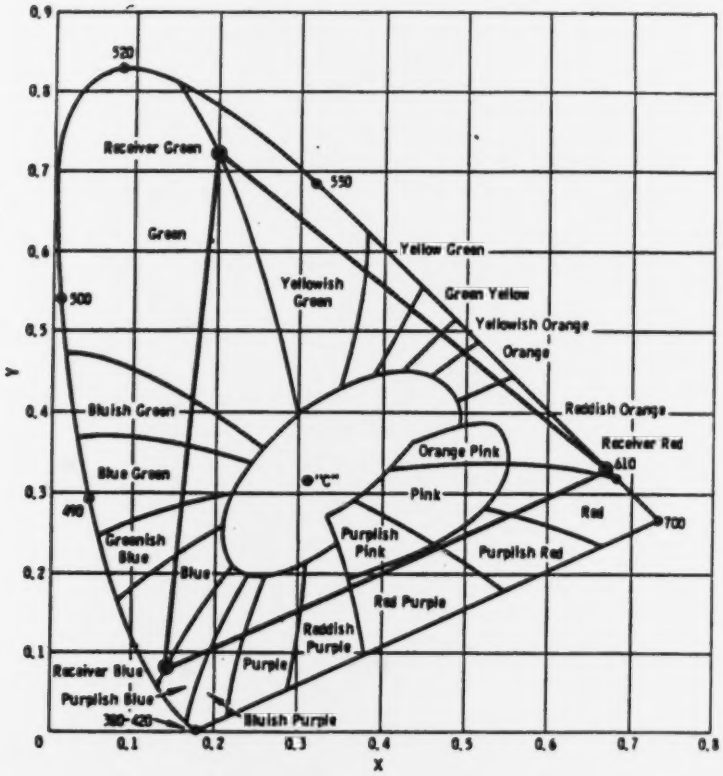


Figure 4.
Triangle of color-receiver primaries plotted in CIE color space.

CHAPTER 5

POLYGONAL OBJECTS

Before discussing coloring polygonal data bases it may be useful to briefly describe what polygonal data-bases are and how they are created. The system for rendering polygons at the Computer Graphics Research Group (CGRG) was developed by Franklin Crow[9]. A simple example of a piece of polygonal data used by this system is a representation of a cube. The data structure that CGRG uses looks like this:

```
data_info       8 points 6 polygons
***************
 8 POINTS     x, y, z
 1:   -0.5   0.5   0.5
 2:   0.5   0.5   0.5
 3:   0.5   -0.5   0.5
 4:   -0.5   -0.5   0.5
 5:   -0.5   0.5   -0.5
 6:   0.5   0.5   -0.5
 7:   0.5   -0.5   -0.5
 8:   -0.5   -0.5   -0.5
***************
6 POLYGONS   # of vertices, actual points
 1:   4   1 2 3 4
 2:   4   8 7 6 5
 3:   4   2 1 5 6
 4:   4   3 2 6 7
 5:   4   4 3 7 8
 6:   4   1 4 8 5
***************
```

This type of data file can be created by either hand entering the x, y, z coordinate values on a computer terminal or by programs developed as tools to help people create these data bases. Such programs can be general enough to allow for the creation of many different types of objects, or can be made specific to an individual object, or class of objects (eg. gears and spirals). Much of the software used for data generation at CGRG was developed by Wayne Carlson[4].

The type of data most commonly used in computer-generated, television commercials (see figure 5) is

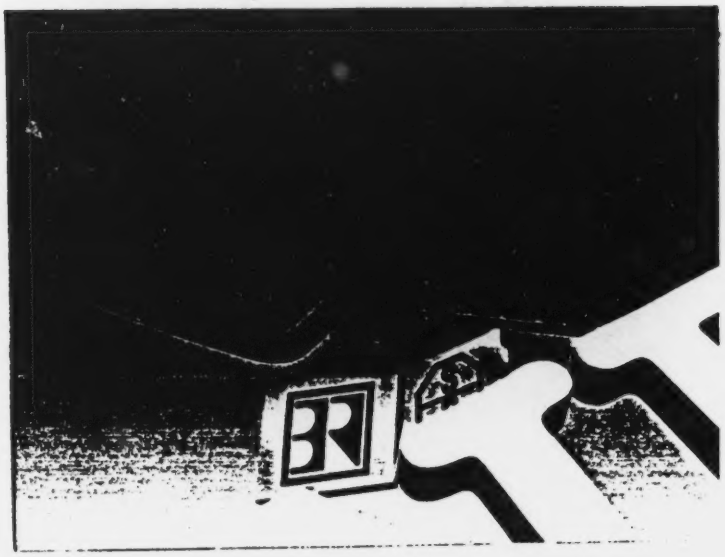


Figure 5.
Flying Logos
Projected data used for
ARD German Television

created by a technique known as projection or extrusion. Here the user works with a bit pad (similar to a drawing tablet) and puck and traces the outline of flat artwork such as a logo or logomark. A computer program then stretches this shape in space creating a three dimensional database with front, back, and side surfaces. A projected square becomes a box. Before the program can "project" the 2d shape, the user has to polygonalize the front surfaces. This is similar to breaking a surface up into puzzle or mosaic pieces. There are certain requirements as to the shape of these pieces. All must be closed, convex, and planar. They must also be traced in a clockwise direction. These restrictions are tailored to the software used to display these shapes.

Another method of creating data (see figure 6) is called solids of revolution or surfaces of revolution. Here the user draws a silhouette edge or profile of a

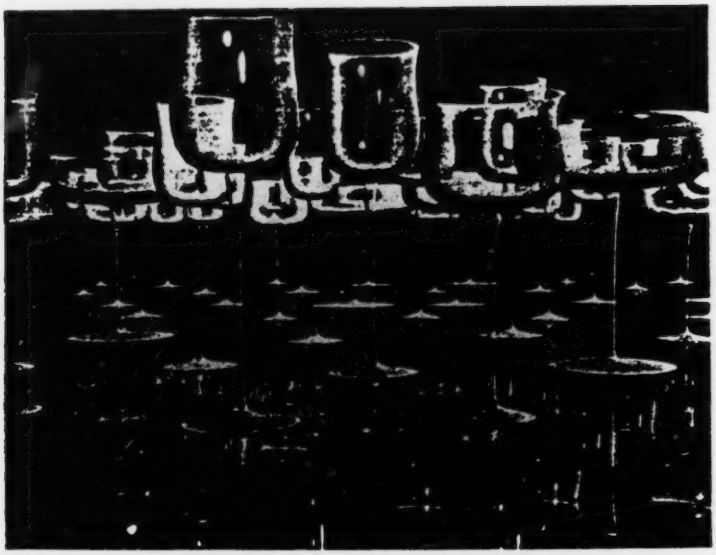


Figure 6.
Wineglasses with Fake Reflections

object (eg. wine glass, bowling pin or vase), and the program revolves this profile around a central axis and defines polygons as it goes around. A sophisticated solids of revolution program will allow the user to create multiple profiles and will interpolate between profiles as it follows its path around the axis. It will also interpolate multiple paths on the vertical axis.

A third method of data generation is known as lofting. This is probably the most general and can be used to create a wide range of shapes, from organic shapes such as a human brain to more architectural objects such as jet airplanes. Here the user traces two-dimensional cross sections or contours of the object to be created. The program then draws polygons between this stack of cross sectional slices. The major drawback of this technique is in the difficulty of creating the original cross sections to be digitized.

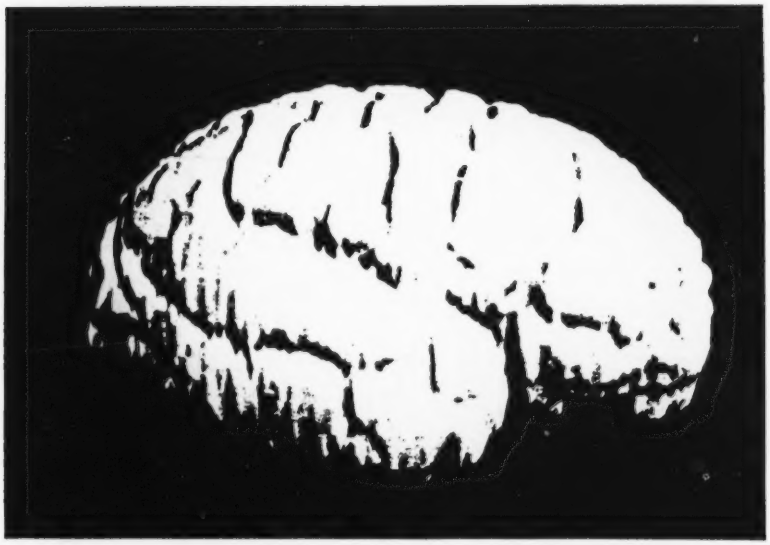
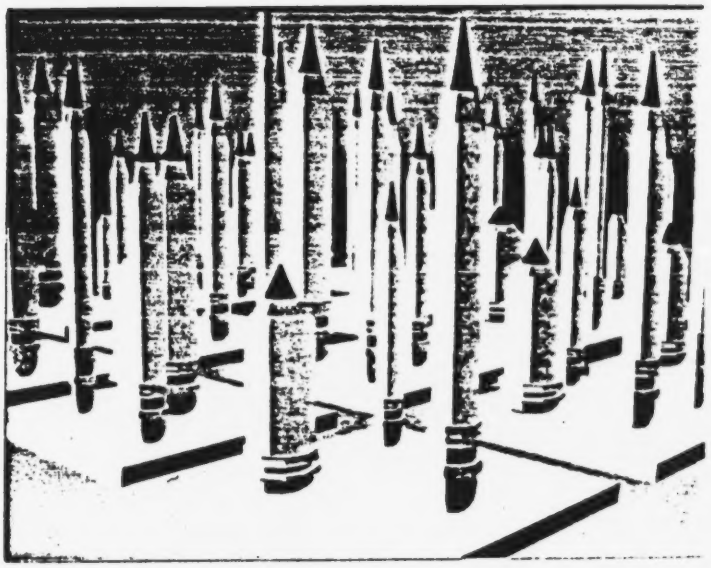


Figure 7.
Don Stredney's and Chris Steele's Brain

Once the initial programs have been used to create basic shapes, other programs can transform these objects by scaling, rotating, and positioning. They can be warped, wrapped, and distorted. Objects can be duplicated, so that a single object becomes a field of objects. They can also be combined with other shapes to create new forms.

The major problem with polygonal data bases is the amount of information needed to be stored and computed for the creation of detailed images. Even a simple shape, such as a ball, can require many hundreds of polygons to simulate its curved surface. With current computer technology it is absurd to imagine a realistic model of nature represented solely as polygons. Ofter types of data such as patches[5], quadric surfaces[6], fractals[12], and newer data types such as particle systems[13], and blobby shapes[3], are needed to simulate the complexities of our environment.



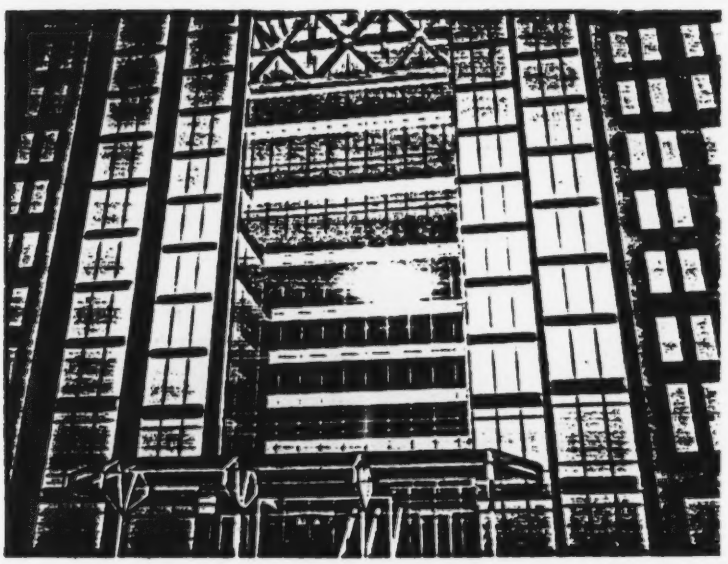


Figure 8.
Duplicating and Combining Polygonal Data

CHAPTER 6

DISPLAY ALGORITHMS

The transformation of polygonal representations of objects in the numerical form to a visual form on a raster screen is accomplished by a program known as a display algorithm or more specificly a scan converter or a tiler. These algorithms can range from simple to complex. The quality of the display algorithm largely determines the overall quality of an image. These display algorithms are in many ways restricted by and tailored to specific display hardware. Programs are written to overcome hardware restrictions, as discussed earlier in this paper. The basis of the display process consists of the shading calculation and the "scan conversion" or moving down the polygon edge and writing values to the screen or frame-buffer. The simplest shading calculation is known as faceted shading. Here the cosine of the angle between the surface normal of the polygon and a line from the polygon to the light source is computed. When the surface of the polygon lies perpendicular to the light source, the cosine is one, and the intensity value is at its maximum. Polygons that are parallel to the light source are assigned the lowest intensity values. Other polygons will be assigned intensities that fall between these extremes.

More sophisticated scan converters create a visual effect know as smooth shading. Smooth shading algorithms average the surface normals of neighboring polygons. A polygon is then shaded with interpolated colors based on the neighboring polygons rather than a single color or shade. The result is an object which looks less faceted. Smooth shading algorithms can vary in the quality of images they can create and calculation time. Smooth shading works on the surface of objects and not on the silhouette edges. These silhouette edges still appear faceted if an insufficient number of polygons are used to describe a shape.

More sophisticated shading algorithms produce an effect known as highlights. The position of the highlight on a object is determined by the position of the eye, the position of the light source and the surface normal of the polygon. The size of the highlight is determined by a value known as the "rolloff factor" and is supplied by the user on an object by object basis.

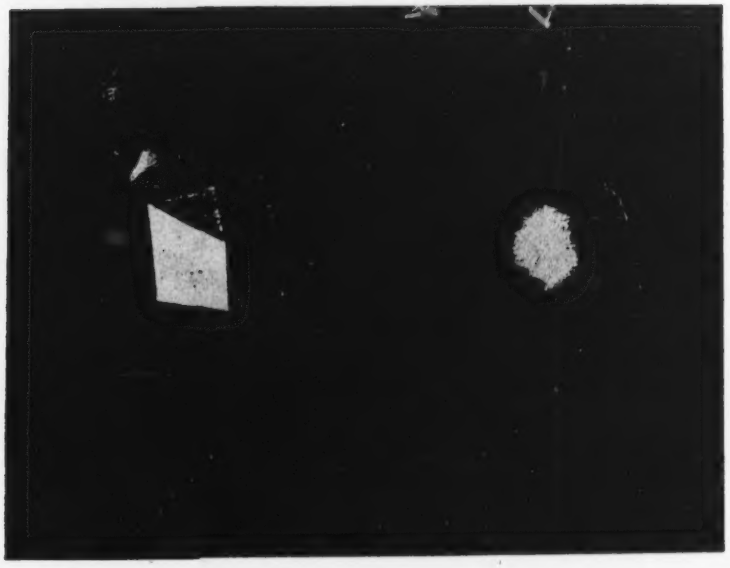


Figure 9.
Faceted and Smooth Shaded Object with Highlight

CHAPTER 7

FRAMEBUFFER COLORMAP CONFIGURATIONS

The scan converter tiles the polygonal shapes into the framebuffer, either polygon by polygon or scanline by scanline. This data, derived from the object description and the shading model has to be related to the idiosyncrasies of the framebuffer. There are two primary methods for doing this, largely dictated by the framebuffer in use. Buffers with less memory are generally designed to utilize a video lookup table with parallel red, green, and blue digital to analog converters. Buffers with larger memories, generally twenty four bits, utilize separate lookup tables for each of RGB components. In the parallel lookup table scheme the data written to the buffer is a single integer value. This value points to a location in the colormap that contains the RGB values that will be assigned to that pixel. When separate RGB intensity ramps are used, three values are written to the buffer for each pixel. Each of these values point to a location in one of three ramps. These ramps are linear intensity scales of the primary colors and correspond directly with the color guns of the display monitor. For example, on a three ramp configuration, gray would be represented by three numbers such as "128 128 128". Full intensity, fully saturated red would look like "255 0 0". In a parallel colormap environment middle gray could be represented by an arbitrary number such as "437". This number would be the "address" to a location in a lookup table where the values "128 128 128" would be stored. The same principle holds true for the red. CGRG uses framebuffers of both types. These where designed and constructed by Marc Howard under the guidance of Dr. Franklin Crow[8].

It should be noted that users of these buffers and color graphics hardware in general need not specify colors using the RGB scheme directly. Algorithms have been developed that enable the user to create colors without dealing with RGB values explicitly. Programs have been written that automatically convert "user friendly" input, such as a descriptive language, into RGB values. One such scheme is based on the color concepts of Hue, Value, and Saturation (HVS). Computer graphic color, whether specified as numbers or words, will eventually be converted by the program into RGB values and written to the display hardware. Users should be given an option to work with

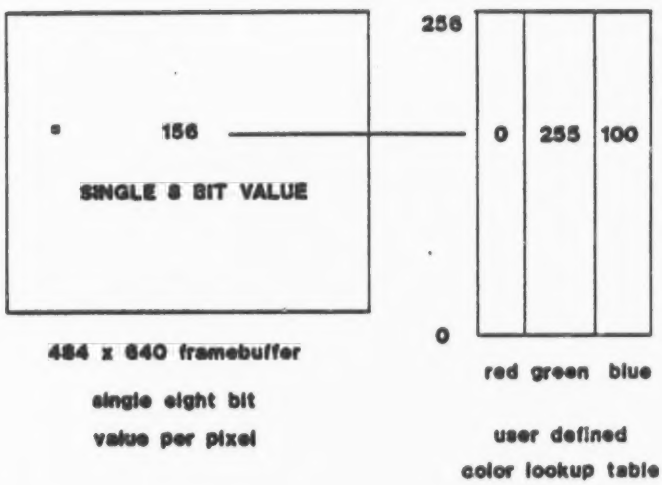


Figure 10.
Parallel Color Lookup Table Configuration

whichever scheme is most natural to them.

Colormaps are user defined and require no particular order or logic. Since colormaps are arranged as arbitrary discrete units, a change in the values in one part of the colormap will not directly influence the other colors on the screen. A single object within an image can change color without any change to the rest of the image simply by writing new values to a particular portion of the colormap. This method can be used to create an inexpensive form of animation that can be produced in real- time. For example, the green leaves on trees in a forest scene can change to their autumn colors without a single frame being recalculated and without change to the color of the other objects in the scene. These changes involve just one write instruction to the colormap which can be accomplished at a rate fast enough to produce realtime animation of a high quality. In an environment using separate ramps, a change in one color can seldom be achieved without affecting the other colors on the screen.

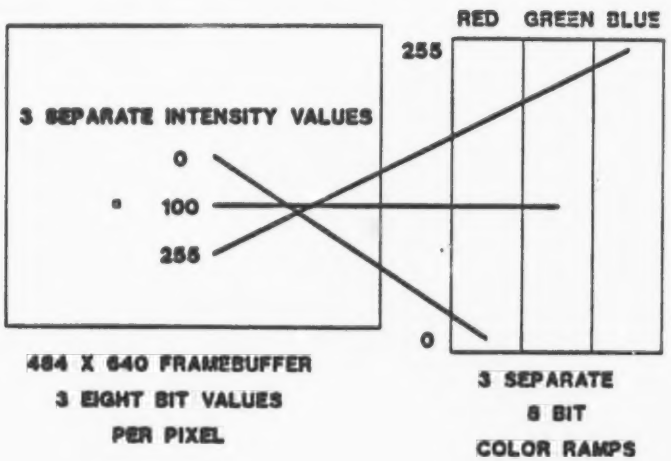


Figure 11.
Full Color Framebuffer Configuration

Colormap animation techniques are missed on systems which utilize separate ramps of red, green, and blue. However the advantages of separate eight bit ramps usually outweigh the advantages of parallel colormap setups. This is due primarily to the tremendous increase in the available colors produced by twenty four bits of color information for each pixel. With over sixteen million colors displayable at any one time, the number of colors on the screen is limited largely by the horizontal and vertical resolution of the image and the imagination of the artist. Organizing and addressing a color lookup table with sixteen million discrete locations would be a difficult task.

The large number of possible color combinations that result from a large bit field can be used to help solve the aliasing problem. Aliasing is a major problem in computer graphics. This refers to frequent jagged edges in computer generated images and to flickering objects in computer graphics animation. This "stair step" effect is especially visible along diagonal or curved edges.

Aliasing must be overcome in order to produce a computer image of high quality. Aliasing has become unacceptable to the computer graphics community. Therefore anti-aliasing techniques (methods of overcoming aliasing) are of significant importance to the artist/user. Anti-aliasing on a three ramp system can be achieved by a concept known as "coverage". Coverage refers to the amount that a particular pixel has been covered by a single polygon or several polygons. This scheme assumes that polygons have been sorted in z depth and will arrive to the framebuffer, via the tiler, in a front to back order.

An image building scenario might go like this: The screen is cleared to the background color and the coverage buffer is set to zero. Polygons are sorted by z depth position. This sorting can be very costly in machine time and can be quite crude depending on the algorithm used. The closest polygon to the eyepoint is tiled into the buffer. As this occurs the coverage for each pixel is

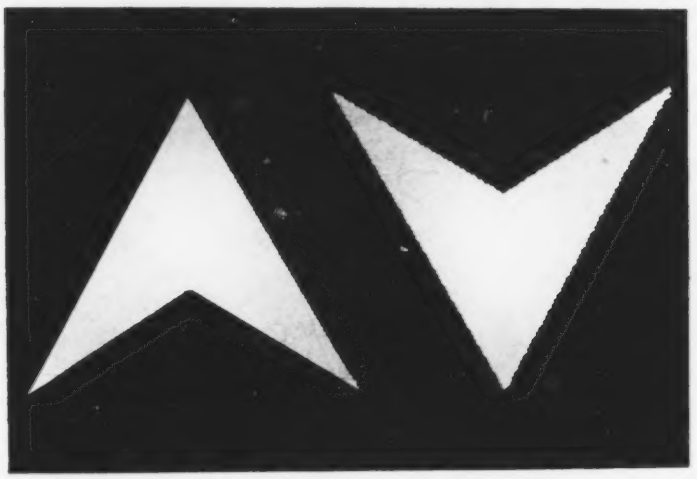


Figure 12.
Anti-aliased and Aliased Edges

computed. If a pixel is fully covered by a polygon it is assigned a value of one. It is a finished pixel and will remain that color for the remainder of the picture building process. This color is a RGB triplet stored as three eight bit values, a separate value for each of the three primaries. If a pixel is only partially covered by a polygon, as in the case of very small detail or on polygon edges, a value less then one is stored in the coverage table. This value is determined by the percentage amount that the pixel is covered.

Before the next polygon is scanned into the buffer, the values in the coverage table are checked for those pixels that are about to be addressed. If the value in the coverage table is zero the initial process is repeated. Covered pixels get the value of one, while partially covered pixels get a fractional value based on the amount that the polygon overlaps the pixel. If the value is one the new color values are ignored. If a fractional value is found in the coverage buffer, the new color values are mixed in with the RGB values stored in the framebuffer to create a weighted average of the two colors. A new coverage value is computed based on the incoming coverage and the coverage already stored in the buffer not to exceed one hundrew percent.

This process is repeated polygon by polygon until all polygons are rendered. As a final step in this process the background color is mixed with the remaining partially covered pixels. Thus anti-aliasing is accomplished between abutting polygons within an object and on its silhouette edges. CGRG's current system uses an eight bit hardware coverage buffer and produces good results in most cases. Improvements could be made either by going to a larger coverage buffer in terms of numerical resolution (i.e. more bits) or by pixel subdivision where the pixel is divided into subpixels each with their own coverage buffer.

Anti-aliasing on a colormap system can be achieved by working with a very select range of colors. The color map could consist of several main colors and then the possible blends of these colors. This is extremely limiting to the animator. Another method is to draw black lines around color boundaries. Then the colors in the colormap consist of a small number of colors that go from black to some full intensity color. These colors will always be mixed with black on the edges so anti-aliasing is achieved by blending only with black. This works well for two dimensional cartoon work but not for rendering objects in the real world.

The coverage scheme has other advantages besides providing for effective anti-aliasing. One of these is that the effect of transparency is easily achieved. Painters have known that to create the illusion of transparency the color of the background objects is mixed with the color of a foreground object to make it appear as if light is transmitted through it. This is done with the coverage scheme by writing out fractional coverage values instead of "ones" when the polygon fully covers a pixel. The color values in these pixel locations will then be mixed with incoming values to create the effect of transparency. The degree of transparency is defined by the user by changing a value known as the "transmittance factor". Using this scheme, various types of transparent materials can be simulated.

Another important feature of this scheme is the ability to mix separately calculated images in the framebuffer. I have developed programs for "post-calculation"

Figure 13.
Transparent Bubbles

image mixing which allows the user to calculate picture elements separately and combine them in the framebuffer. These programs rely on coverage for their ability to composite picture elements with complete anti-aliasing. This software has tremendous utility in a commercial animation environment.

Another advantage of separate RGB ramps is the ability to have an unlimited range of colors within a single object and have a smooth blend between these colors. This effect is known as "vertex coloring". With vertex coloring each point in an object is assigned a color. As the polygon is tiled the color is interpolated between the vertices that describe a particular polygon. For instance, if one point of a polygon is red and the other is blue the color would linearly change from red to blue, passing through magenta, as it is tiled into the framebuffer.

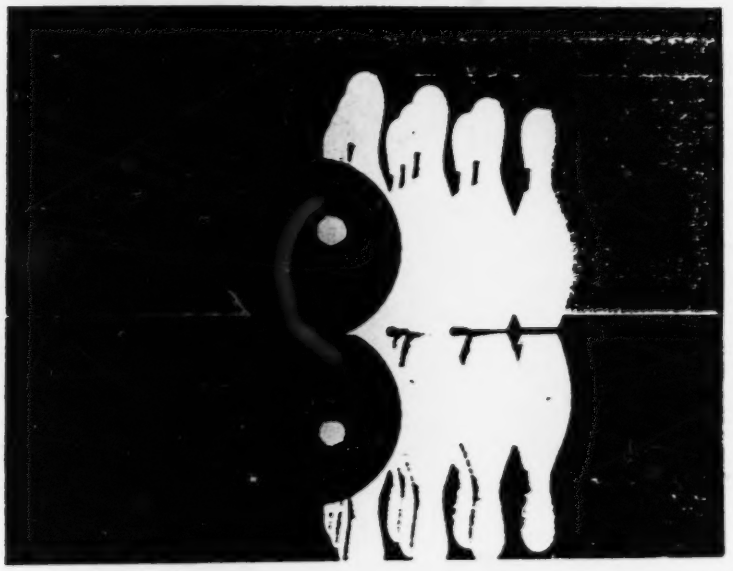


Figure 14.
Bowling Pins
Fake shadow and reflection
created with compositing techniques.

Colormap systems allow the user to give each polygon a separate color from a limited selection of color. However vertex blending is largely impossible, due to the limited range of color.

Before continuing the discussion of vertex color files and polygon color files more attention needs to be given to colormaps. When making the switch from a colormap system to a RGB ramp configuration the quality of images can be increased dramatically. This is due to improved anti-aliasing, transparency, and vertex coloring. However the advantages of colormaps should not be disregarded.

A working knowledge of colormaps can greatly enhance the software tools typically found in a graphics environment which relies on separate RGB ramps. Examples of some of these benefits include: a method for coloring objects based on non-monochromatic color ramps, a decrease in image generation time resulting from a drastic cut in framebuffer I/O, a significant decrease in disk space needed to store frames of animation, and the ability to change color before and after an image has been calculated.

In much commercial animation, objects are often displayed by themselves or calculated separately and then combined in the framebuffer to create a composite image. Because of this, in a large number of cases, only eight bits of color information need be associated with each pixel until it is displayed on the screen for final recording. While it is being calculated, written to the buffer, read from the buffer, stored on disk, loaded on tape, etc., the size of each pixel element need only be sixteen bits. In this case eight are used for intensity and eight for coverage, instead of our current use of thirty-two bits. The use of sixteen bits instead of thirty-two will result in real cuts in the overall costs of producing animation. The saving of time and storage space is very significant. The estimated time required to calculate a frame in a particular animation sequence is a major factor in determining its cost. A major part of this time is spent reading and writing pixels values into and out of files and framebuffers. If the size of this picture information (data) is cut in half, the time needed to create and store an image is reduced by a significant factor. Also the limited resource of disk storage space could be utilized more efficiently resulting in increased capacity and a decrease in time and money spent on tape archival.

Not only are these savings obtained, but other advantages also result. One possibility is a type of color file that can broaden the range of color associated with an object. In many cases a piece of data is calculated with a single color specified for that object. For example, an object may be colored full intensity yellow. As it is tiled into the buffer the object will be shaded from a linear palette ranging from low intensity to high intensity yellow. This type of shading does give an object a three dimensional look but, it is limiting from an artistic point of view. It does not simulate real-world objects well.

In a colormap environment it is possible to create shaded objects with subtle hue changes throughout the intensity ramp. This is an excellent approach for simulating various types of material without expensive mathematical calculations. By cross-fading between these non-linear, polychromatic color ramps effects such as metallic reflections can be suggested.

David Warn [14] describes an interactive program with which the user can dynamically blend an object color with a color for the specular highlight to simulate different materials or paints under variable lighting conditions. This is done by replacing the video lookup table values and does not require that the image be recomputed. He states that this approach requires a parallel lookup table and cannot be used with three, eight bit lookup tables. However, a similar system can be developed on a three ramp configuration. The difference is in the way the program is implemented. Color can easily be mixed in realtime on a three ramp configuration.

This is a type of three-dimensional painting. A calculated image of an object that has been originally colored white (RGB of equal intensities) is painted interactively. The three color ramps are treated as a single unit. The result is a pseudo color lookup table with 256 locations, like that of an eight bit framebuffer. Colors can be created that vary in hue, saturation, and value as they transverse the intensity ramp.

This approach gives the artist a broader range of color possibilities in addition to making the job of mixing colors faster and more controlled. The output of this interactive session is then stored in a file. Then, when the display algorithm scan converts this object, it can look up the color associated with a particular intensity and write the corresponding red, green, and blue values into the framebuffer. This file can also be used while

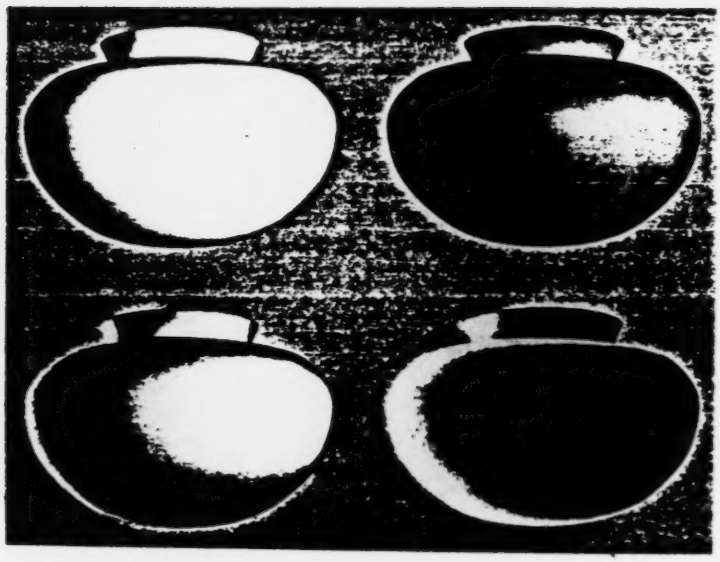


Figure 15.
Colormap Changes
A pre-calculated image is made to look different by modifying colormap during display

displaying a precalculated sixteen bit image. By looking up the values in this file a sixteen bit image can be expanded to a full color image for final recording.

Another approach is to calculate and store an object with eight bits of intensity information, as mentioned above. Then during the recording process load this eight bit intensity value into the separate bytes that lookup to the red, green, and blue ramps. Then, by simply changing the values in these ramps the object can be recorded many times in any number of colors. This is a very useful for doing work such as television logos where the client wants several versions of the same move in different colors.

CHAPTER 8

POLYGON COLORS AND VERTEX COLORS

As mentioned earlier, a system that utilizes separate lookup tables allows for the implementation of "vertex coloring". A unique color can be associated with each coordinate point that makes up the object. The color is blended between vertices as it is scanned into the framebuffer. This scheme also allows for "polygon coloring" where each polygon is assigned a separate color.

There is no blending between hues with polygon coloring. Blending of intensities does occur between polygons in the case of smooth shading, but each polygon has a unique hue and saturation. The borders between polygons are clearly visible if adjacent polygons are not assigned the same color value, even in the case of smooth shading. With vertex coloring, polygon edges are blended by a method similar to smooth shading, however, this is done independent of the intensity (shading) calculation. These linearly interpolated color blends are multiplied by the intensity value when determining the color for each pixel. Both faceted and smooth shaded objects can be colored with either polygon coloring or with vertex blending. On our system, transparency values are included within vertex and polygon color files. Transparency is interpolated just as are red, green, and blue.

The effects of vertex coloring and polygon coloring look quite different and as such serve diverse purposes. Polygon coloring is useful when a clear definition of color and pattern is desired. Checkerboard grids, tile floors, building windows, and striped objects, provide examples where polygon colors are appropriate. The object will have clean, hard edges, between abutting polygons.

Vertex coloring produces a soft blended transition between the colors of adjoining vertices. The maximum number of colors per single polygon is limited only by the number of points allowed per polygon by the display algorithm in use. These colors can be located anywhere in the RGB color space. Smooth blends occur between colors located closer to each other in color space. Other factors affecting the color blend are the size, shape, orientation and number of points in any group of adjoining polygons which share vertices, and the size of the

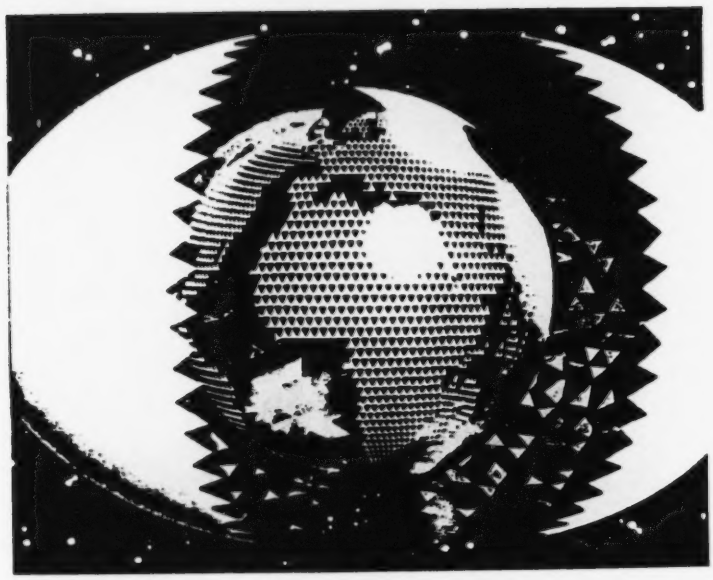


Figure 16.
"Rebirth"
Polygon coloring used to
create color patterns

object on the screen. Vertex coloring has many uses, such as land forms, body parts, logos, and simulations of metal and other types of reflective surfaces.

The animator is able to create a large number of effects with the same piece of data through exploration of these coloring methods. The form of an object is only one factor in determining the overall affect it has in a composition. The same piece of data can be used to simulate grass, mountains, water, and clouds. This is accomplished by using different color files, as well as by changing the object's scale, image resolution, and the lighting and viewing parameters in the scene.

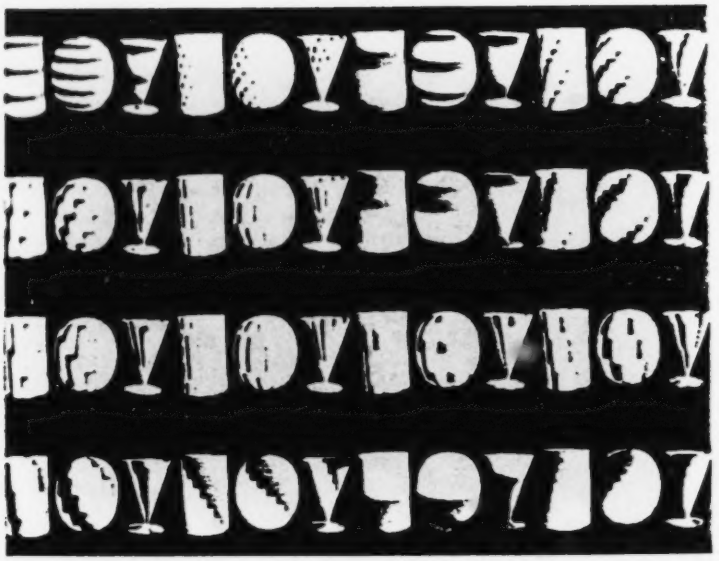


Figure 17.
"Looped Vertex Colors on Solids of Revolution"

CHAPTER 9

CREATING COLOR FILES

The ability of coloring objects with polygon and vertex files creates a need for programs to create these files. These files are known as vertex color (.vcl) and polygon color (.pcl) files on our system. One of the most simple and useful programs I have written to create these files is called 1 col. This program produces color files based on the concept of "nested looping". The user inputs a pattern of colors that will be repeated until their number equals the number of points or polygons in a particular data file. The user specifies the number of unique colors and the number of repeats for each of these colors. This color file, consisting of a repeated pattern made up of unique colors each with a specificed number of repeats, will correspond directly to the data file with which it will be associated with. Therefore the arrangement or structure of the points and polygons in the data file will be directly related to the organization of the colors that appear on the screen when the object is displayed.

A simple example of the program's use and of how its output relates to a corresponding data file is as follows: To generate a "pcl" file that will give a polygonal grid a checkerboard appearance, two colors are input. Each of these colors are repeated once per pattern. This pattern of two colors is output until the number of polygons in the grid has been reached. If the grid has an odd number of polygons in each of its rows a checkerboard effect will be achieved. However if the grid has an even number of polygons per row a striped pattern will be created instead (see figure 18).

It is necessary for the user to know the structure of the data file when working with this particular program. This structure will directly influence the effect created by the associated color file. Therefore the data should be created to accommodate the desired effect. For instance, if stripes are desired a grid should be created with an even number of polygons per row. A grid with an odd number of polygons would be needed for a checkerboard.

A program such as 1 col works well with data objects generated algorithmically. These data files tend to be regular in their internal organization. Coloring objects

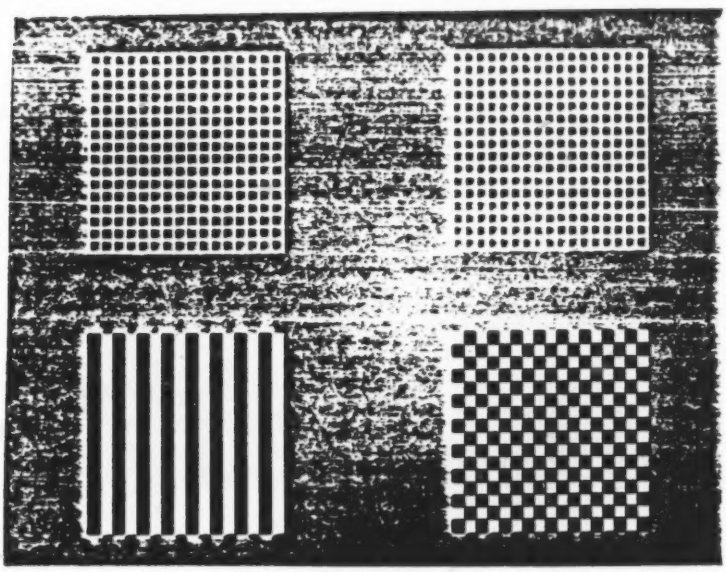


Figure 18.
Strips vs Checkerboard Effect

generated by "solids of revolution" software is an ideal application for a program of this nature, due to the uniformity of their data structure. The number and arrangement of the polygons in these objects will have a direct effect on the image that appears on the screen. This structure will be a product of the number of points that make up the profile and the number of points used to describe the path. The number of points in the profile will be the major factor in determining the color patterns that will result because the points and polygons are output vertically, profile by profile. The number of points in the path will determine how visible the seam is between the beginning and end of the color file.

These examples (see figure 19) clarify how the data structure will help determine the pattern of color created on the object. They graphically depict how different number of points in a profile will affect the visual outcome when associated with a color file from 1 col. The horizontal axis represents the number of points in a

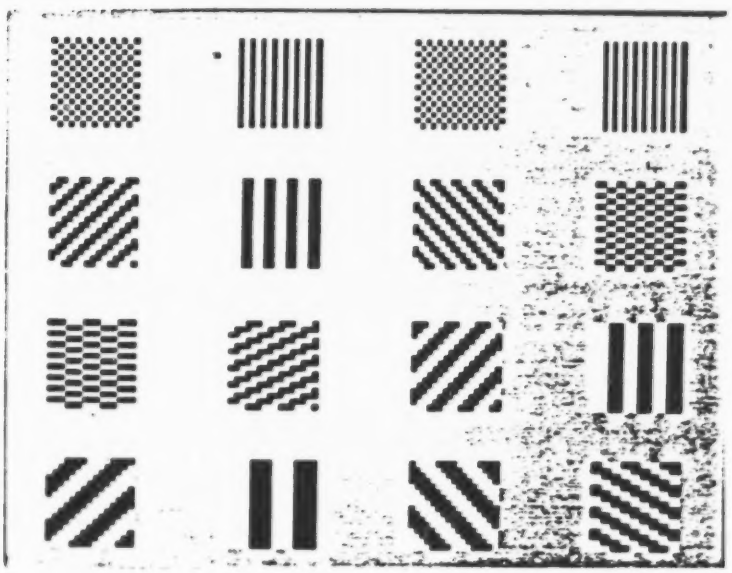


Figure 19.
Polygonal Structure vs Polygon Color Pattern

profile used to describe a shape, ranging from fifteen to eighteen in this example. These numbers either refer to polygons or points. An object with seventeen points in a profile will have sixteen polygons down its side. Therefore the same input file to 1 col will create different patterns when used to output vertex or polygon color files (see figure 20). The number of repeats for each of the two colors being used, ranging from one to four are on the vertical axis. Each of these rectangles represents the color pattern that will result given that particular combination of points on a profile and the number of repeats per color. The diagonal patterns that develop will appear as spirals on objects created by solids of revolution. These spirals will be very angular in the case of polygon colors and somewhat smooth when vertex coloring is used. The blend of these spirals can be made smooth by interpolating the colors that make out the pattern themselves. For example: The input file to 1 col for a spiral based on three repeats of two alterating colors and a profile of multiples of seventeen points

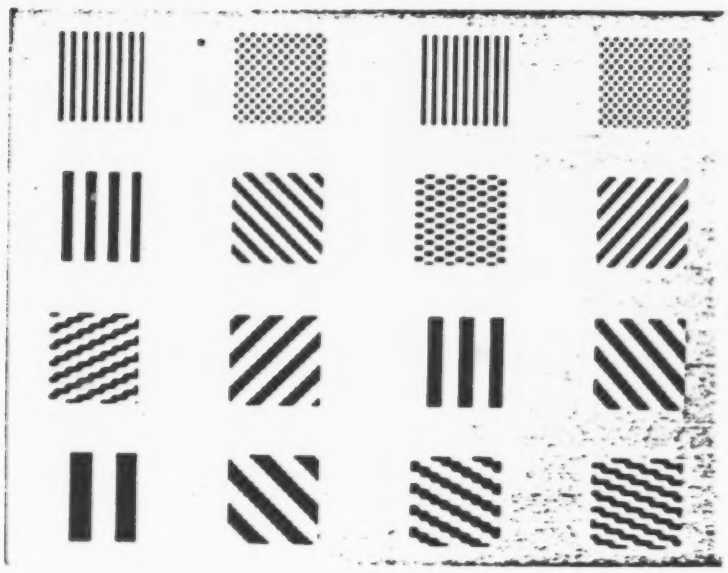


Figure 20.
Polygonal Structure vs Vertex Color Pattern

(17,34,51,etc) would look like this. The spiral that results would have very irregular edges because of the distance between the two in color space.

```
**************************************
number
of colors
2
**************************************
number
of repeats      red   green   blue  transparency
3                1      1      1        0
3                0      0      0        0
```

If a smooth blend is desired the stairstep between colors can be reduced by a technique analogous to anti-aliasing. The colors that makeup the pattern should be interpolated themselves. For example:

```
**********************************
number of colors
6
**********************************
number
of repeats  red    green    blue    transparency
1           .66      .66      .66        0
1           1.     1.       1.         0
1           .66      .66      .66        0
1           .33      .33     .33        0
1           0.       0.       0.         0
1           .33     .33     .33        0
```

Another useful set of programs I wrote are named blend poly and blend colors The former outputs "pcl" files, the later "vcl". These programs take an input file that is a list of color values that are to be mapped onto

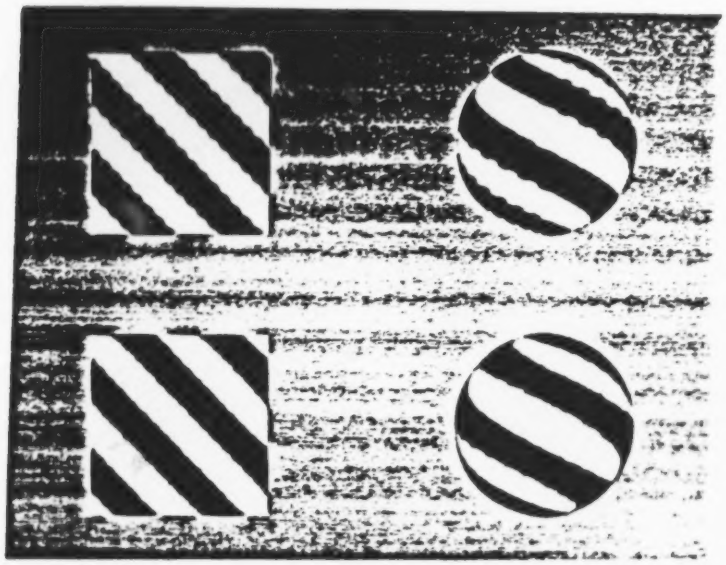


Figure 21.
Smoothing Vertex Color Blending

the object based on its bounding box. A bounding box is an imaginary box inside of which the object will prefectly fit. This program works as follows: an input file is associated with a particular axis, (eg. the depth position axis, z axis). The object is broken up into regions along this axis. These divisions are based on the number of colors in the list. That is, the number of regions will be one less then the number of colors in the file. In the case of vertex colors, the program goes through the data file and locates each point in terms of these regions. A point that falls on the most negative edge of the depth axis will be assigned the color described by the first line of the file. A point that falls near the center of the z bounding box will be associated with the color described by the middle line in the file, etc. The program will output the color that is associated with each point or will blend between the two colors that define that region, depending on the needs of the user. In the case of polygon colors the program will compute the centroid of each polygon and color that polygon based on the location of that centroid.

The output of these programs is not influenced by the object's file structure, as is 1 col. They work in world space, therefore the order and number of points and polygons within the data file itself will have no affect on the outcome of the program. The advantage of this is that the user does not need to be concerned with how the numbers look inside the data file. The knowledge of the object need only be visual, or how it looks in wire frame mode.

This program is aware of the data structure in terms of how polygons are arranged in space. A blend cannot occur between three colors if the object only has two points lying on an axis. Therefore the user needs to be sensitive to how the object looks in world space. He needs to be concerned with how evenly the polygons are arranged and that there are enough points in the object to relate to the colors that are to be mapped. Oddly spaced and shaped polygons will result in an uneven blend of color.

A program such as blend col has numerous uses in an animation environment. It is especially useful for objects made by projection. Objects, such as logos can have color that is be blended across the front, either horizontally or vertically or across the sides. This can add a great deal of interest to an object and can be used to simulate many qualities. Blend_poly is good for lofted logos with beveled edges or for polygons which are to be colored based on their orientation in space.

Figure 22.
Landscape with Vertex Coloring

Blend col is excellent for coloring fractal landscapes. Lakes and mountain peaks can be created by coloring points based on their elevation. The best results occur when the colors go from dark saturated colors at low elevations to lighter unsaturated colors at higher elevations. To suggest depth, another vertex color file should be created based on the z axis. Points closer to the observer should be saturated and have brighter intensities. Colors in the distance should approach middle gray. These two vertex files should be combined to create a new file.

Color files can be mixed by several methods, each of which will produce different results. The most straightforward method is to add the color numbers together and divide the result by the number of files being mixed. This method tends to push the colors towards middle gray because a high intensity color mixed with a low intensity color will produce a color of middle intensity. A variation on this method takes a weighted average of the

various intensities. For instance a third of one color will be mixed with two thirds of another. This process can be animated to create smooth transitions between one color file to another, much like a film cross dissolve. This is good for simulating reflections. Another method multiplies the two files together. This will tend to lower the object's intensity because one fraction multipled by another produces a product that is less then either of the two fractions. All methods are useful as long as the user understands how the programs work and what results are likely to be achieved.

Programs such as these for combining color files to create new color files allow the artist a wide range of color possibilities. Don Stredney used this approach when coloring the iris of the human eye. He used 1 col to give the object a striped appearance and used blend col to make it darker towards the center. These files were averaged to create the final color file.

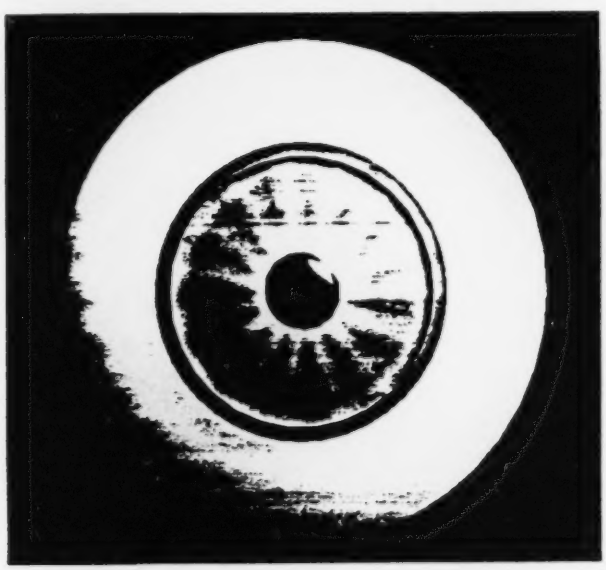


Figure 23.
Don Stredney's Eyeball

CHAPTER 10

CONCLUSION

I have described some of the factors that affect color in computer generated images and have described a few programs I have written to apply color to polygonal data bases. The material presented here addresses itself to a small fraction of the information pertaining to color and computer graphics. Many pertinent issues were not discussed. These include colored lights sources, ambient light, texture mapping, and the use of alterative display algorithms for creating color on shaded images.

Colored light sources have a major affect on the object's hue and value. In the real world when a green light is cast on a red object the object appears to be black. Color light sources reproduce this effect by multiplying the RGB values of a light source with the colors (RGB values) assigned to the object. Thus a red light on a white object will make the object appear red and a red light on a green object will make the object appear black. Generally colored lights are not painted with fully saturated colors but are some hue mixed with white. An object's highlight color is generally the same as the color of the light, however on some systems this can be controlled independently by the user.

Ambient light will also affect the color of an object as it appears on the screen, determining the lowest (dark) end of the object's value scale. Ambient light is a multiplication factor used to shade polygons facing away from the light source. An ambient light factor of zero will produce an object that is bright on the side of the light, while tapering off to black on its other. This type of lighting effect produces a stark dramatic look. A typical ambient light factor is .3, this simulates the affect of "fill" or "bounce" light in a scene. An ambient light set to 1. creates a very artificial effect often described as "candy apple" because objects tend to look like hard candy. The affect of ambient light can be enhanced when colormaps are used because the nonilluminated areas can be shaded with any color rather than simply darkening the object's original hue and saturation.

Texture mapping can be used to create a wide range of color possibilities not possible with vertex and polygon

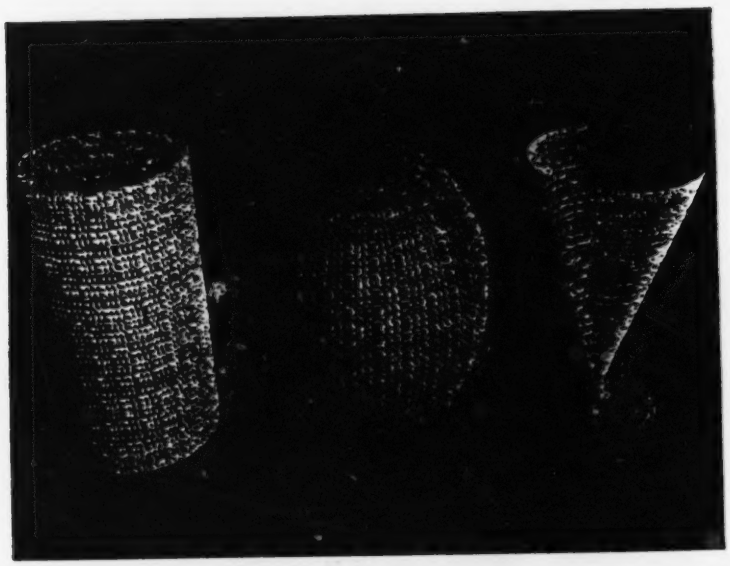


Figure 24.
Texture Mapped Images

coloring. Texture mapping enables the user to overlay color on an object more or less independently of its polygonal structure. An image of a tree can be mapped onto a polygonal representation of a ball. The polygonal mesh of the ball has nothing to do with the shape of the tree, however, the image of the tree needs to be mapped onto the ball, polygon by polygon. Mapping images onto the polygons can be quite difficult especially if the object is not easily "unfolded" or broken down into a duplicatable pattern. Texture mapping is generally used to enhance the shape of an object, that is, to make a model look more like the thing it is supposed to represent. For instance a camera digitized image of a beer label could be texture-mapped onto a digital model of a can. Texture mapping can also be used to make a simple shape appear complex, for instance, a rectangle can be made to look like a piece of wood. Images used for texture-mapping can be created by digital paint programs, camera digitized images, or computer generated scenes. Textures can also be generated algorithmically during the display process.

Texture mapped objects are shaded with the same calculation used for other polygonal models.

Shaun Ho has developed many programs that allow the user to color objects with effects not easily obtainable with the methods mentioned above. One such program, used to create metallic looking objects, colors the object based on a conceptual spherical environment in which the model is placed. As the object moves within this multicolored environment the color is "reflected off" the sphere and onto the object. The sphere itself is never seen, just the effect of its color on the object. This result in bands of color that move across the object as it is animated. This effect can also be simulated by manipulating an object's color map or vertex color file. However, the motion of the color will not be directly related to the object's movement and can look artificial if not done well.

Figure 25.
Metallic Effect

Another program of Ho's redefines the polygonal structure of an object based on the laws of refraction and then texture maps a flat scene onto this new object. This results in objects that take on the appearance of crystal. The development of programs such as these broaden the range of images that are possible to be produced on a digital system. They offer new challenges for the user to work with color to create the best possible image within the limits of the display hardware and software.

Developing the ability to make the best possible image with the existing resources is a difficult task. Every phase of process is time consuming, from data generation to image calculation. Generally, before a program or technique is mastered to the point of subtle control, a new program comes along with another set of problems to solve.

People making images with computers generally are not visual artists. They are people developing a new technology and their images are a by-product of this process. Theirs is an exploration in technological capability, and not artistic expression. The evaluation of computer imagery is generally based upon software capability. The precision, the difficulty, and the newness of the algorithm used to create an image largely determines the image's merit within the computer graphics community. Issues of form, color and concept are all but ignored. Color is a matter of practicality and not subtlety. Those interested in subtlety are frustrated by the limits of the display hardware. No two monitors look alike and most of the color detail is lost during transmission or photographic reproduction.

As the field develops in terms of sophistication and availability, the use of color will increase in importance. Software will tend to stabilize, giving artists the opportunity to perfect their ability to deal with the subtle nuances of color within the limits of the system they are using. Once a program has been written by those leading this new technology, it will generally be up to the user to realize its ultimate potential. Hopefully the impact of an image, with its subtle nuances and artistic style will become more important than its algorithmic complexity.

List of References

1. Albers, Josef. Interaction of Color. Yale University Press, New Haven, 1975.

2. Birren, Faber. Color, Form and Space. Reinhold Publishing Corporaation, New York, 1961.

3. Blinn, James F.. "A Generalization of Algebraic Surface Drawing". ACM Transactions on Graphics 11, 3 (1982), pp. 192-198.

4. Carlson, Wayne E.. "An Advanced Data Generation System for Use in Complex Object Synthesis". Proc Graphics Interface '82 (May 1982), pp. 197-204.

5. Carlson, Wayne E.. "An Algorithm and Data Structure for 3D Object Synthesis Using Surface Patch Intersections". Computer Graphics (July 1982). to appear..

6. Catmull, Edwin E.. "A Subdivision Algorithm for Computer Display of Curved Surfaces". UTEC-CSc-74-133. Salt Lake City, Utah, December 1974. University of Utah Dept of Comp Science.

7. Catmull, Edwin E.. "A Tutorial on Compensation Tables". Computer Graphics 13, 2 (August 1979), pp. 270-285.

8. Crow, F. C. and Howard, M. W.. "A Frame Buffer System with Enhances Functionality". Computer Graphics 15, 3 (August 1981), pp. 63-69.

9. Crow, F. C.. "A More Flexible Image Generation Environment". Computer Graphics 16, 3 (July 1982), pp. 9-18.

10. Ennes, H. E.. Television Broadcasting Camera Chains. Bobbs-Merrill, New York, 1971.

11. Newman, William M. and Sproull, Robert F.. Principles of Interactive Computer Graphics, 2nd ed.. McGraw-Hill, New York, 1979.

12. Norton, Alan. "Generation and Display of Geometric Fractals in 3-D". Computer Graphics 16, 3 (July 1982), pp. 61-67.

13. Reeves, William T.. "Particle Systems-A Technique for Modeling a Class of Fuzzy Objects". Computer Graphics 17, 3 (July 1983), pp. 359-376.

14. Warn, David R.. "Lighting Controls for Synthetic Images". Computer Graphics 17, 3 (July 1983), pp. 13-21.